Methodology to Develop Domain Specific Modeling Languages

Subhrojyoti Roy Chaudhuri
Swaminathan Natarajan
Amar Banerjee
Tata Consultancy Services, Research & Innovations
India

Venkatesh Choppella
venkatesh.choppella@iiit.ac.in
International Institute of Information Technology,
Hyderabad
India

Abstract

Domain Specific Modeling Languages (DSML) significantly improve productivity in designing Computer Based System (CBS), by enabling them to be modeled at higher levels of abstraction. It is common for large and complex systems with distributed teams, to use DSMLs, to express and communicate designs of such systems uniformly, using a common language. DSMLs enable domain experts, with no or minimal software development background, to model solutions, using the language and terminologies used in their respective domains. Although, there are already a number of DSMLs available for modeling CBSs, their need is felt strongly across multiple domains, which still are not well supported with DSMLs.

Developing a new DSML, however, is non trivial, as it requires (a) significant knowledge about the domain for which the DSML needs to be developed, as well as (b) skills to create new languages. In the current practice, DSMLs are developed by experts, who have substantial understanding of the domain of interest and strong background in computer science. One of the many challenges in the development of DSMLs, is the collection of domain knowledge and its utilization, based on which the abstract syntax, the backbone of the DSML is defined. There is a clear gap in the current state of art and practice, with respect to overcoming this challenge.

We propose a methodology, which makes it easier for people with different backgrounds such as domain experts, solution architects, to contribute towards defining the abstract syntax of the DSML. The methodology outlines a set of steps to systematically capture knowledge about the domain of interest, and use that to arrive at the abstract syntax of the DSML. The key contribution of our work is in abstracting a CBS from a domain into a Domain Specific Machine, embodied in domain specific concepts. The methodology outlines, how the Domain Specific Machine, when coupled with guidelines from current practices of developing DSMLs, results in the definition of the abstract syntax of the intended DSML. We discuss our methodology in detail, in this paper.

CCS Concepts • **Software and its engineering → Domain specific languages**.

Keywords domain specific modeling language, domain specific language, language engineering, modeling

ACM Reference Format:
Subhrojyoti Roy Chaudhuri, Swaminathan Natarajan, Amar Banerjee, and Venkatesh Choppella. 2019. Methodology to Develop Domain Specific Modeling Languages. In *Proceedings of the 17th ACM SIGPLAN International Workshop on Domain-Specific Modeling (DSM '19), October 20, 2019, Athens, Greece.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3358501.3361235

1 Introduction

Domain Specific Languages (DSL) can have a wide range of applications, as pointed out by Nordstrom et al.[22]. Domain Specific Modeling Languages are special type of DSLs, developed primarily to model Computer Based Systems (CBS). Examples of CBSs are software, hardware, databases, documents and procedures. DSMLs significantly simplify, modeling complex CBSs. The main purpose of DSMLs is to empower domain experts, who lack software programming knowledge and skills, to model CBSs using the concepts and terminologies of the domains, they understand well. The ability to model such systems using DSML, provides the benefit of improving productivity and efficiency towards their engineering. The models created using the language, enable us to express systems using formal notations and allow to reason about them mathematically. They also help to communicate system descriptions precisely.

The need and scope of a DSML must first be established, and these determine its design. Figuring out the scope involves, structuring information about the area of application of the DSML and building understanding and knowledge about systems in that area. The area of application of a DSML

DSM '19, October 20, 2019, Athens, Greece

ACM ISBN 978-1-4503-6984-8/19/10…$15.00
https://doi.org/10.1145/3358501.3361235

implies its domain. Structuring information about the application area is known as *Domain Analysis*, also discussed in 2. As pointed out by Mernik et al.[20], DSML development can be hard since it requires both domain knowledge as well as language development knowledge and skills. One of the key initial steps in the development of DSML for the identified domain, is to define the abstract syntax of the language. An abstract syntax is the data structure which is independent of any particular representation and has the primary goal to store information about the system being modeled using the DSML. Based on the abstract syntax, the subsequent step is to define the concrete syntax of the DSML. The concrete syntax of a DSML defines the specifics of how elements of the abstract syntax are displayed using the language and edited. Defining the concrete syntax is again a challenging task as it is influenced by aspects which are not purely technical such as aesthetics, look and feel and so on.

One of the critical challenges towards DSML development, is the problem of analyzing the domain specifically with a view to defining the abstract syntax of the DSML. Although, domain analysis is an established area of research, there is little knowledge available that provides insight into how a domain should be analyzed specifically with the view to defining the abstract syntax of a DSML. Focusing specifically towards mitigating this challenge, we have come up with a methodology that enables arriving at the DSML abstract syntax systematically for the selected domain. Our methodology does not address the problem of defining the concrete syntax of the DSML even though it is an important area of concern and we plan to address it in our subsequent work.

Our methodology applies to developing DSMLs, whose abstract syntax are defined using meta models. We developed this methodology while implementing DSMLs for multiple domains, more specifically, supervisory control systems for a variety of distributed systems, discussed in section 3. A key idea behind our methodology is in abstracting a Computer Based System for the selected domain, into a Domain Specific Machine (DSM). A DSM abstracts systems in a domain into explicated reference architecture designs, embodying the key domain concepts. Systems abstracted and represented as DSMs, make it easy to identify the key elements of the meta model of the DSML. We discuss how the available guidelines from existing literature for developing DSMLs, when coupled with the DSM, leads to defining the DSML meta model in a systematic manner.

In this paper, we discuss this methodology in detail with supervisory control systems used as an example domain to illustrate our methodology. The paper is organized as follows: 2 reviews the relevant literature that helps in developing DSMLs, 3 provides a high level description on the fundamentals of our methodology, 4 illustrates the methodology, 5 provides a discussion on the applicability of the approach for different types of CBSs, followed by conclusions.

2 DSL Design Methodologies - review

This section provides a summary of the existing literature, that provide insights into methodologies used towards developing DSLs and DSMLs.

Need for DSML Sprinkle et al. point out that the need for a DSML should be carefully analyzed [29]. Only if the domain is found to be wide enough with considerable number of users trying to solve similar problems of repetitive nature, it makes sense to develop DSMLs. Mernik et al., Sprinkle et al. provide a checklist for identifying domains that may be well suited for developing DSLs [20, 29]. Language Oriented Programming [6] is already an established argument for building DSLs for software development and has inspired multiple DSLs[3].

Stakeholders and Requirements of DSML Determining the scope of the DSML is an important activity, that needs to be carried out after domain selection. Scoping allows us to decide more precisely what parts of a domain and its associated concerns the DSML requires to be able to model. Kolovos et al. [17] throws light on how to arrive at such requirements. According to them, the three typical stakeholders for DSMLs are a) system or software engineers, who are responsible for choosing and using a DSML to model a system, b) customers for providing feedback, and c) developers, who are responsible for constructing and managing DSMLs. Based on the objective of these stakeholders, the DSML requirements can be formulated. Kolovos et al. also provides a list of requirements generically applicable to developing DSLs [17]. Many of the requirements for DSLs are derived from the principles of building general purpose languages (GPL).

Types of DSMLs Fowler, Gill, Selic point out the types of DSMLs that are possible to develop [10, 12, 28]. DSMLs can be developed from scratch as completely new languages, which lead to creation of external DSLs. DSMLs can also be created by embedding new domain specific constructs in a host language, which are popularly known as internal DSLs as well as embedded DSLs (EDSL). Internal DSLs can have multiple types such as shallow and deep embedding as pointed out by Gill. Selic suggest that creating DSMLs from scratch can be daunting and extending an existing language such as UML[27] maybe an easier approach. Selic provides an approach to enhance UML to build DSMLs [28].

Meta Model A key constituent towards defining a DSML is the domain model that provides crucial inputs towards defining the language meta model as pointed out by Lédeczi et al. in [18]. The problem of creating domain model is itself an area of research popularly known as *domain analysis*. A prerequisite to designing a DSML is detailed analysis and structuring of the application domain knowledge [14, 30]. Although, most of the literature do point out that the meta model for a DSML comes from domain knowledge, there are

only a limited set of systematic approaches available currently to arrive at the same from domain knowledge. Čeh et al. suggests an ontology based approach to domain analysis to aid DSL design. Van Deursen and Klint provides an approach to derive the language model from the feature description of domain specific systems, using a textual language used in conjunction with UML. Pescador and de Lara points out an approach that allows to define the initial meta model of the DSL by capturing requirements into DSL-Maps and using meta model design patterns.

Hudak points out that although there have been hundreds of DSLs developed over the years, systematic approach towards their study has only started more recently [15].

Guidelines and Patterns Karsai et al. provide detailed guidelines on developing DSMLs which is a comprehensive list of points to keep in mind while developing DSMLs [16]. The guidelines, as expected, are abstract rather than concrete steps. [10, 20] provides detailed insight into the patterns that can be put together towards the development of DSMLs. Mernik et al. provide high level patterns related to deciding whether a DSL is needed, how to analyzing the domain, how to design and implement DSLs and so on. The patterns are at higher levels of abstractions, listed in tables. López-Fernández et al., Pescador et al. point out how common patterns can be used to develop language meta model by involving domain experts [19, 24, 25]. Patwari et al. provide experience from developing a DSL for control systems [23].

Implementation Implementation of DSLs are very well supported by a plethora of language workbenches as pointed out by Erdweg et al.. We implemented our approach using XText [8], a popular DSL development workbench. Such workbenches are useful for creating external DSLs from scratch. DSMLs can also be implemented by embedding them in a host language, as discussed above.

3 Methodology Foundations

This section serves as a prelude on the foundations, based on which our methodology is developed. It establishes the key ideas behind our methodology to develop DSMLs, with supervisory control systems used as an example area of application. The section starts with a brief introduction on supervisory control systems and current approach towards their engineering and then moves on to laying the foundations of our approach.

3.1 Supervisory Control Systems

A supervisory control system, an example chosen to illustrate our work, solves the problem of controlling and monitoring the execution of a wide variety of complex systems such as radio telescopes, robotics based industrial automation systems, smart environments and many more. It implements the important orchestration and error handling logic for all such complex systems and also addresses concerns related to safety, security and other aspects for its host system. Control systems are an important class of components required for building a variety of complex systems with vastly different goals as pointed above. We use the term control system to refer to supervisory control systems in the rest of the paper.

Control Systems Engineering Approach Based on our experience from building control systems for apparatuses of different types, such as fusion reactors, radio telescopes and IoT based solutions, we saw that the approach to engineering control systems is significantly similar, even if the systems for which they are built differ vastly in their engineering. The approach to create the architecture of control systems, identifying their requirements and design patterns, have a lot of commonality across all these systems. However, many such design details are worked out from the ground up, in a repetitive manner across projects. This is partly attributable to the non availability of modeling languages that are specifically meant for modeling control systems using common concepts and vocabulary from the control systems domain. This implies that control systems make for a good candidate to build a DSML to model them.

3.2 Conceptual Overview

The foundation of our approach is motivated by our insight that the abstract syntax of a DSML consists of a a)Domain Specific Machine, an abstract machine which defines the computational schema for the space of interest b)domain vocabulary and c)linguistic structures. Our approach points out, that development of abstract syntax for a DSML involves a) developing it's Domain Specific Machine (DSM) b) working out the declarative expressive elements, in terms of the vocabulary of concepts in the domain, and their interrelationships. This enables to define the structure of the computational space, and c) working out the linguistic structures from the language definition point of view, needed to express the computational processes required over those elements. Thus designing the abstract syntax of a DSML starts with generating domain understanding into a Domain Specific Machine definition and working out subsequently, the declarative constructs and linguistic structures. This becomes the essential input to develop the concrete syntax of the DSML subsequently, but is not addressed by our methodology as pointed out earlier.

Domain Specific Machine The primary source of inputs required to develop a DSML is, knowledge about the domain and CBS in the domain. Important questions that drive the collection of this knowledge are: what are the key knowledge elements that serve as the primary input for the DSML meta model? How to acquire and organize this knowledge? How can this knowledge be utilized to define the DSML meta model? We answer these questions with the key idea of a Domain Specific Machine (DSM). Although it is inspired

by the concept of abstract machines, commonly used in automata theory [5] in computer science, it serves a different purpose. Turing machines, statemachines are examples of abstract machines which serve as foundations for analyzing computer programs.

A DSM enables representing CBS of a particular class, such as control systems, in terms of an abstract architecture pattern expressed using the concepts and vocabulary of that class. We use the term *Domain* to mean such classes in this paper. Control systems are often represented by reference architectures [1, 26]. We observe that, even if control systems are developed from scratch for most complex systems, they reuse common patterns, implicitly. A typical hierarchical control system consists of controllers arranged in a hierarchy, with the controllers communicating with each other through exchanging commands, events or alarms, and data. Such communication provides inputs for a controller to trigger execution of an action, as well as changing its own behavioural state. The entities in such architecture patterns, e.g. controllers in the case of control systems, may express their own internal working logic in terms of an abstract machine such as a statemachine. A statemachine allows to capture the behaviour of a controller, such as its state transitions while performing an action, based on the reception of an input which could be a command, event or an alarm. A reference architecture, formulated and represented as a collection of entities with mutual relations, incorporated with domain specific terminologies and their working logic expressed as abstract machines such as statemachines, is a DSM as can be seen in Figure 4.

In order to develop a DSM, knowledge about the domain is gathered, in a manner that enables to express the DSM using the terminologies used in the domain. This Knowledge can be organized in terms of a domain vocabulary comprising of a) entities used towards engineering the CBS, control systems, in our example, b) functions and processes that the CBS implements c) communication elements used by the CBS and its internal components d) semantics of internal working logic expressed in terms of abstract machines. Once a DSM is constructed for a domain, it provides the primary reference to derive the meta model of the DSML to model systems in the domain. The DSM, used in conjunction with the guiding principles discussed next, results in formulating the DSML meta model.

Guiding principles for defining DSMLs For a DSML to be effective, its meta model definition should comply with a set of guiding principles defined from the linguistics perspective. These principles are derived by studying the current guidelines available in literature towards developing DSLs, e.g. the ones proposed by Karsai et al.[16].

Declarative Support declarative specifications using domain vocabulary to model the system, e.g. control systems. This is needed for modeling the system for a specific context and to instantiate its engineering entities and specifying assignments, relations, behaviour and so on. For control systems, examples of domain vocabulary are commands, responses, alarms, events, which are its communication elements. The vocabulary includes system components or entities such as parent and child controllers and their states, domain functions such as alarm handling, sending and receiving commands and responses, managing resources and so on. A DSML to model control systems for a specific context, should enable specifying details about the control system, using elements from this vocabulary.

Structural Constructs Incorporate appropriate structural constructs based on which the entities or building blocks of the control system can be composed to model the end to end system. Based on literature, there is no clear evidence as to how the language structures are arrived at. For example, in case of object oriented languages, concepts such as 'class', 'functions' and so on have been defined keeping in mind the principles of object oriented software programming[21]. However, there is no clear understanding available as to how these constructs were arrived at. In case of the SysML modeling language, its meta model is influenced by the principles of systems engineering[11]. Structural constructs play a vital role in defining the concrete syntax of the DSML as well.

Behaviour Provide mechanisms to specify how the CBS reacts to changes and incorporates its own internal working logic. The means to specify such details, should not require low level programming skills as also pointed out by Karsai et al.[16]. The SysML modeling language provides a list of useful diagrams such as activity, scenario, statemachines, interactions and so on to model the working logic of a system, however, they lack domain specificity to specify the details, unless they are extended.

4 Methodology

In this section, we discuss our methodology for building DSML for control systems in steps as depicted in Figure 1, based on the foundations discussed in section 3. The primary objective of the methodology is to derive the required meta model for the DSML, as pointed out in section 3.

4.1 Capture Domain Knowledge - Step 1

This is the first step, in which knowledge about the domain, control systems in our case, is captured systematically. In our methodology, we capture this knowledge using the tools and concepts of model driven engineering. However, there can be other approaches used to capture the same knowledge as well, such as ontology capturing tools[4], feature modeling[30], mind maps and so on. The approach chosen should be simple enough so that it can be used by domain experts and customers without background in software engineering to explicate their domain knowledge using the

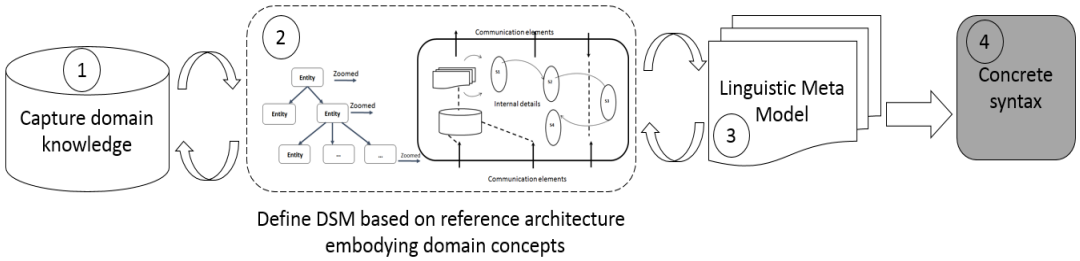

Figure 1. Pictorial view of the proposed methodology

approach. It is the job of the system engineer and DSML developer to select the appropriate approach and define the required knowledge schema using meta modeling or other tools.

We use a meta model as shown in Figure 2, to capture the domain knowledge. This meta model is conceptualized from the general theory of systems engineering and does have similarity to the meta model used by the SysML modeling language. The sole purpose of this meta model is to capture domain knowledge and this should not be confused with the DSML meta model which is the final output produced by using the proposed methodology.

As can be seen from Figure 3, the meta model helps to capture knowledge about the control systems domain such as its stakeholder roles, their objectives and the set of functions that come together to achieve those goals. Examples of stakeholders in the domain of control systems are control system engineers and operators. One of the goals of the operator is to ensure correct usage of the system, achieved through functionalities such as validating commands, handling alarms and so on, as shown in Figure 3.

Collect Domain Vocabulary Carefully analyzing the knowledge using the steps explained in 4.1, it is observed that they provide significant hints to derive the required vocabulary for the selected domain, control systems in this case. The collected vocabulary can be categorized as communication elements, domain entities, functions and modules and their interrelations as pointed out in section 3. The inputs to derive the vocabulary and their categorization primarily come from domain experts with the help of systems engineers.

4.2 Define the Domain Specific Machine - Step 2

In this step, architecture patterns that are commonly used to build CBSs in a domain, are gathered. This is a key input to define the DSM. From the gathered architecture patterns, select the pattern of choice that the DSML needs to support. This activity is primarily carried out by systems engineers with inputs from domain experts and participation from DSML developers, if required. The selected architecture pattern is carefully mapped with the domain knowledge captured in the previous step. This helps to express the architecture pattern in terms of the domain specific vocabulary. For example, in case of control systems, a commonly used architecture pattern is hierarchically composed controllers as pointed out by Roy Chaudhuri et al. [26]. The architecture is represented as a hierarchy of controllers communicating with each other through exchanging commands, responses, events and alarms. The controllers incorporate mechanisms to express their reactions to change. The change could be triggered by a command, event or alarm apart from other possible sources. The reaction itself could be about changing the state of the controller or executing an action or cascading the trigger through a series of commands invocations or events and alarms generation. Abstract machines such as statemachines and transition systems[9] can be used to express such reactions or behaviour of the entities in the DSM. This enables to reason about the model, created using the DSML mathematically. As can be seen from the Figure 4, a control system can be made up of recursive controllers arranged in a hierarchy, with each controller implementing its own statemachine.

4.3 Identify and Define the Linguistic Structure of the DSML - Step 3

In this step, the key structures that the DSML needs to support are identified based on the DSM. We propose that the potential language structure should be carefully analyzed using the DSM and the structuring decisions should be backed by good rationale. The end users such as customers and domain experts should play a significant role in this activity guided by DSML developers who have good understanding of best practices for developing DSLs. There can be multiple factors influencing the identification of structural constructs as discussed below:-

Identify interfaces In this step, the DSM is analyzed to identify the entities, that require providing information about their usage through interface definitions. For example, in the domain of control systems, a controller requires to provide its interface description, so that engineers of other controllers can understand how to interface with it. Such descriptions

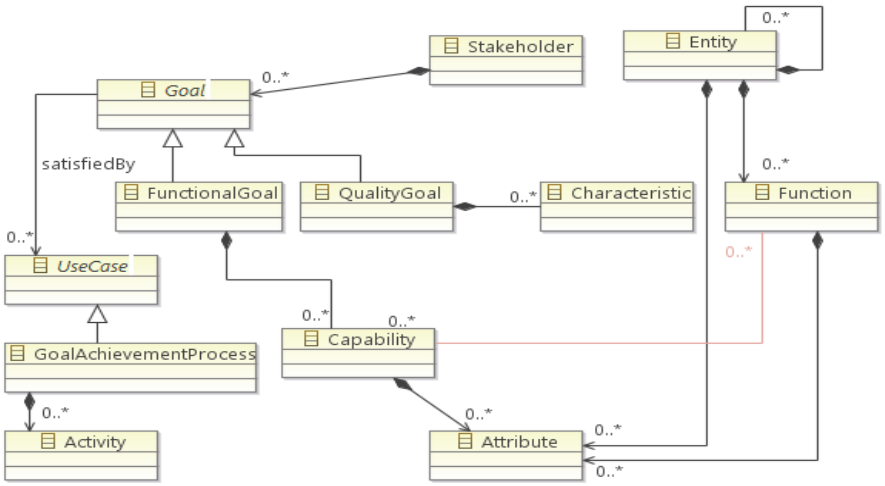

Figure 2. Systems meta model to capture domain knowledge for control systems

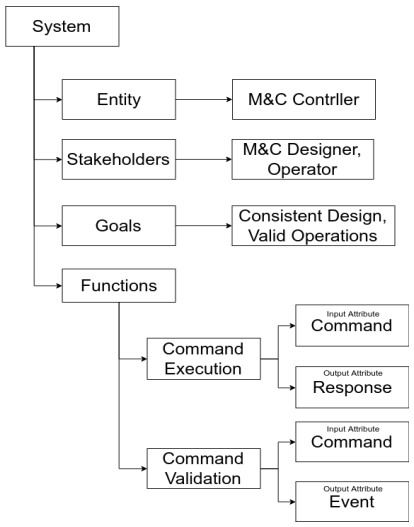

Figure 3. Control systems domain knowledge captured using the Systems meta model schema

capture details about the interaction elements of the controllers and information about their status. This is a standard practice, we have observed based on our experience from building control systems. The required interface is derived by looking at the functions owned by entities, their input and output parameters, as indicated by the DSM. Karsai et al. point out the need for DSMLs to support interface specifications in their paper [16]. This need can be judged best by the end user or customer.

Structures from Domain Functions One way to identify the language structures could be done by directly deriving them per domain function, inline with the approach used to create DSML using Haskell [12]. The domain knowledge captured using the systems meta model, as outlined by our approach is the best source to identify the domain functions. The DSML can have a structural construct per domain function, e.g. a construct corresponding to the domain function alarm handling for control systems. This, however, might lead to large number of structural constructs, which might cause confusion in their usage. This is addressed by modularizing the DSML, with DSM serving as the reference to point out scopes for modularization. This is discussed in the paragraph 4.3.

Modularity An approach to mitigate too many structural constructs derived from large number of domain functions is to group them appropriately, to make the DSML modular, inline with the guidelines about modularity by Karsai et al.[16]. This helps to keep the language simple and easy to use. Multiple factors can influence the need for groups such as grouping all the content related to a particular entity or their activities. Modularization can also be done by collecting together all the content related to a particular concern e.g. security, in one place. Language designer must choose a basis for modularity primarily based on the inputs from the customer, such that it leads to maximum clarity and usability of the DSML. For example, domain entities highlighted in the DSM, implementing specific set of domain functions, can help identify a group. In case of control systems, a group can be created using the controller entity, which is responsible for implementing a specific set of domain functions. Within each entity group, there can be multiple subgroups created, for example, grouping functions with common characteristics such as common inputs and outputs. For control systems,

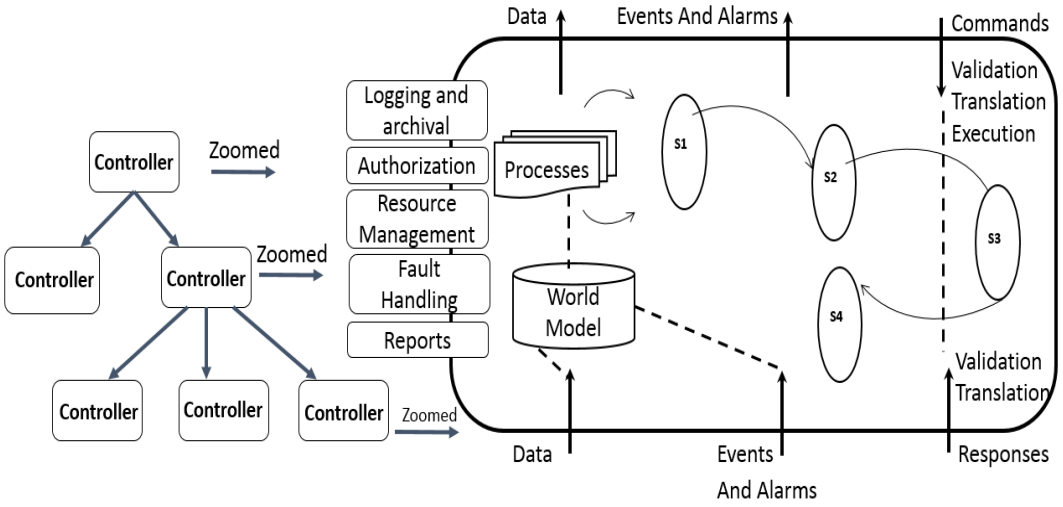

Figure 4. A typical architectural pattern employed in the domain of control systems

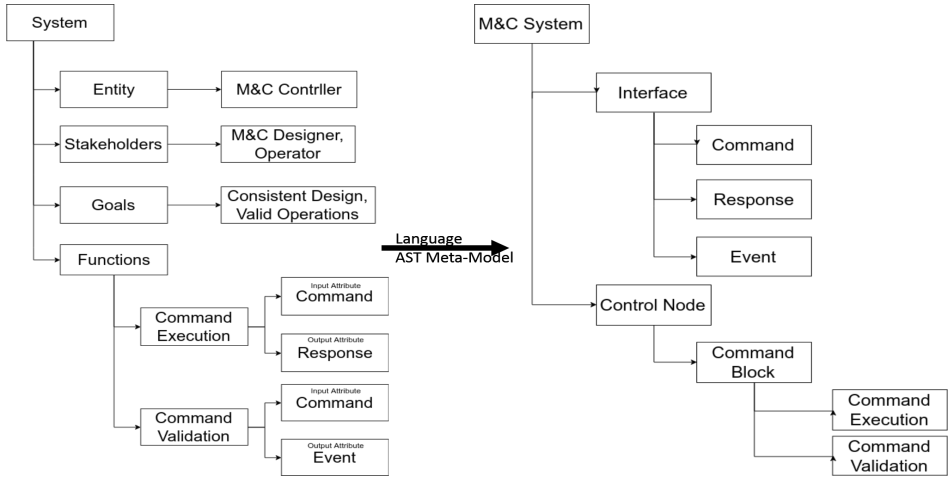

Figure 5. Meta model of the DSML derived based on the steps discussed so far

a controller can further group its functions based on its responsibilities related to commands, alarms, events and so on. Customer inputs are vital to arrive at such decisions for grouping.

Common Concerns A key input to defining the scope of a DSML is identification of different cross cutting aspects, applicable to the DSM. In the domain of control systems, the aspects could be security, safety, reliability and other quality concerns. Much of such aspects are cross cutting in nature and can be considered to be domains in their own right. The role of such aspects should be carefully analyzed and their specification needs should be identified by referring the DSM and the vocabulary. The analysis can be carried out primarily by systems engineers with inputs from domain experts. Based on this analysis, the structural constructs of the DSML may be enhanced, only if such specification footprints are small. However, we believe that instead of incorporating support for modeling all such aspects using one DSML, each aspect should be treated as a separate domain, in case they have large specification footprints. The same methodology that we propose, can be used to define the meta models for the DSMLs for those domains. This leaves with a an important question about how will the collection of such DSMLs inter operate, for interdependent domains? Although this is still an open question we have not addressed in our methodology, we believe that the area of aspect oriented programming [13] can provide hints towards solving this question. Aspect weaving is a common technique used in the area of aspect oriented programming and principles from this area could

help answer some of the questions [13]. However, the usage of aspect weaving towards DSML development is still an open area of research.

The Figure 5, shows an example of a meta model for a control system DSML, that is derived using the steps discussed so far. The meta model represents the abstract syntax of the DSML.

Specification of the Internal Logic A DSML needs to enable specifying how the entities in the DSM react to changes. Since the DSM builds on abstract machines such as state machines, it already provides hints about the entities, which are amenable for such specifications. By carefully looking at the DSM and the identified structural constructs of the language created using the previous steps of the methodology, the appropriate placeholders for such specifications can be identified. One way to implement such specification mechanisms, is by inlining them within the identified structural constructs. For control systems, a DSML construct corresponding to a domain function, e.g. command Handling, may support specifying state transitions within itself. The need for specifications to capture reactions, should be analyzed and the right structure where they should belong in the DSML should be identified as part of this step. This can be worked out by the DSML developer with inputs from domain experts.

4.4 Concrete Syntax - Step 4

Concrete syntax covers not only choice of keywords, but also decisions as to whether to use semicolons, infix or postfix notations for expressions. Such decisions are governed by usability and aesthetic choices. Although this is an important area of concern, the proposed methodology does not address creation of the concrete syntax for the DSML. Selection of the syntactic keywords should be primarily carried out by domain experts or customers with the help of DSML developers. However, Karsai et al.[16] recommends to keep the keywords in the DSML, inline with the domain concepts. Syntactic sugar in the concrete syntax may also include visual or graphical elements apart from textual keywords. Our methodology has not delved into the problem of designing the concrete syntax of textual or graphics based DSMLs.

5 Discussion

Designing DSML has so far been looked at as an art, or perhaps a craft, since there are guidelines, approaches and tools to facilitate it, but not quite a systematic theoretically motivated methodology. Based on our experiences with creating DSMLs for control systems as well as some other domains including robotics, and examination of popular DSMLs in various domains, we have proposed a methodology for DSML design that includes:-

- a meta model to collect the concepts and keywords in the domain, needed for DSML design
- a Domain Specific Machine that captures the computational architecture for the particular domain, often existent in the form of a domain reference architectures
- working out the linguistic structure based on the entities, interfaces and patterns in the architecture, and modularity choices based on typical domain practices, usability considerations and existing guidelines on DSL development.

The primary input for defining the scope of the DSML comes not only from the understanding of the functional outcomes of systems in domains, but also associated concerns such as security and choices among alternative implementation mechanisms. The reference architecture depicted as the DSM, needs to include patterns to address these concerns, and this flows into the design of the linguistic structure.

While we have illustrated this approach using the example of the supervisory control systems domain, we believe that this applies to other domains as well. For example, spreadsheets are DSLs, which follow an architectural pattern of cell grids linked by derivational relationships. Spreadsheets help to specify the derivation mappings among cells. Relational databases have a structural architecture of tables with fields linked by keys, and an associated computational architecture for querying and updating the database which drives the design of Structured Query Language (SQL). Figure 6. illustrates the sequence of steps involved in mapping from the database problem scope to domain meta model to computational architecture to SQL, the DSML for the domain. Hypertext Markup Language (HTML) is an interesting case, where the language design is driven by a model of the meta information associated with documents, that is, document structural elements, formatting information, embedded objects, hyperlinks; and architectural concepts. For example, that a document consists of sections divided into paragraphs, that the meta information is scoped hierarchically and so on. In this case, the DSM is a web browser whose computational processes are not covered by HTML, since its scope is limited to a description of the structure of documents.
The applicability of our methodology and conceptual approach to a wide range of problem domains, gives us confidence that this systematic methodology can be applied to the design of DSMLs for any domain. In addition to control systems, we have also applied this approach to the design of DSMLs for robotics and business workflow specifications.

6 Conclusion

Domain Specific Modeling Languages significantly ease describing solutions at higher levels of abstraction. They empower domain specialists, devoid of software programming skills to model Computer Based Systems using the vocabulary of their domain. However, building DSMLs is non trivial and is carried out as a one off exercise in the current practice.

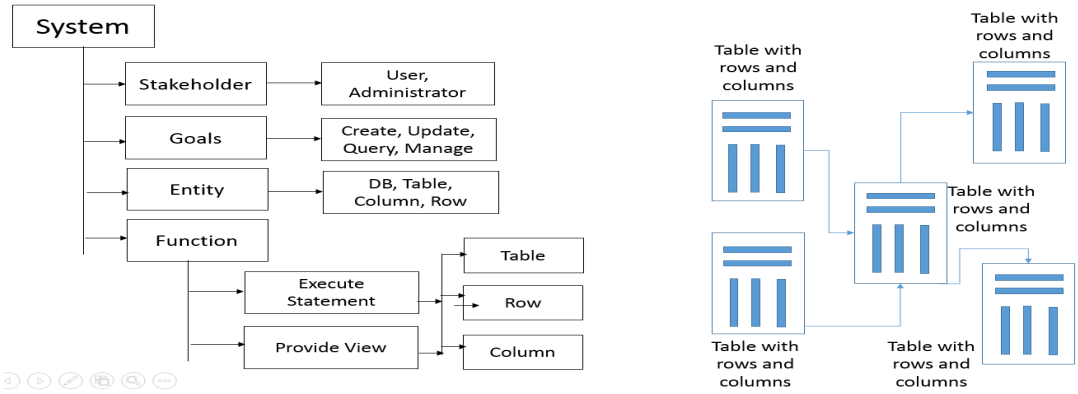

Figure 6. Knowledge from the domain of relational databases

It is not only effort intensive, but also requires significant knowledge and exposure to language development, to do a good job of building a DSML for a new domain.

There are many challenges to developing a DSML. One of them is the challenge of defining the abstract syntax of the DSML. In this paper, we have discussed our methodology that attempts to mitigate this challenge, more specifically for the DSMLs whose abstract syntax can be defined using meta models. The key idea of our methodology is in capturing and organizing knowledge about the intended domain, using the concept of a Domain Specific Machine. The DSM and the guidelines in the existing literature, provide enough hints to derive the meta model of the DSML for the domain of interest. However, the approach is still conceptual without formalization and rigor. The approach also needs to be implemented manually by DSML developers based on their judgment. The methodology is meant to serve as a guiding framework to define the meta mode of a DSML and is not developed to be a process which can be automated. Final decisions about the selection of the right elements for the DSML meta model, is left to the judgment of the DSML developer. The approach does not address other important aspects of DSML development such as what should be its concrete syntax, should it be textual or graphical and so on.

The methodology is a reflection of the thought process that we ourselves followed, while developing DSMLs for multiple domains. Our DSMLs have been used by multiple large scale international projects for modeling control systems. Based on the successful application of the thought process, we explicate it into a methodology so that it benefits the DSML development community in general. We also used our experience from developing this methodology to contribute towards the requirement of next generation systems modeling languages, as part of multiple international forums known for defining standards in the area of object oriented programming and developing modeling languages for systems engineering. Our next steps are to a) formalize the concept of Domain Specific Machine and explore how the methodology b) can be supported with tools and technologies and c) can be extended to address defining concrete syntax of DSMLs.

References

[1] James S Albus. 2002. 4D/RCS: a reference model architecture for intelligent unmanned ground vehicles. In *Unmanned Ground Vehicle Technology IV*, Vol. 4715. International Society for Optics and Photonics, 303–310.

[2] Ines Čeh, Matej Črepinšek, Tomaž Kosar, and Marjan Mernik. 2011. Ontology driven development of domain-specific languages. *Computer Science and Information Systems* 8, 2 (2011), 317–342.

[3] Ryan Culpepper, Matthias Felleisen, Matthew Flatt, and Shriram Krishnamurthi. [n. d.]. From Macros to DSLs: The Evolution of Racket. ([n. d.]).

[4] Stefan Decker, Prasenjit Mitra, and Sergey Melnik. 2000. Framework for the semantic Web: an RDF tutorial. *IEEE Internet Computing* 4, 6 (2000), 68–73.

[5] Stephan Diehl, Pieter Hartel, and Peter Sestoft. 2000. Abstract machines for programming language implementation. *Future Generation Computer Systems* 16, 7 (2000), 739–751.

[6] Sergey Dmitriev. 2004. Language oriented programming: The next programming paradigm. *JetBrains onBoard* 1, 2 (2004), 1–13.

[7] Sebastian Erdweg, Tijs Van Der Storm, Markus Völter, Meinte Boersma, Remi Bosman, William R Cook, Albert Gerritsen, Angelo Hulshout, Steven Kelly, Alex Loh, et al. 2013. The state of the art in language workbenches. In *International Conference on Software Language Engineering*. Springer, 197–217.

[8] Moritz Eysholdt and Heiko Behrens. 2010. Xtext: implement your language faster than the quick and dirty way. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*. ACM, 307–309.

[9] A. Finkel and Ph. Schnoebelen. 2001. Well-structured transition systems everywhere! *Theoretical Computer Science* 256, 1 (2001), 63 – 92. https://doi.org/10.1016/S0304-3975(00)00102-X ISS.

[10] Martin Fowler. 2010. *Domain-specific languages*. Pearson Education.

[11] Sanford Friedenthal, Alan Moore, and Rick Steiner. 2014. *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann.

[12] Andy Gill. 2014. Domain-specific languages and code synthesis using Haskell. *Commun. ACM* 57, 6 (2014), 42–49.

[13] Jeff Gray, Ted Bapty, Sandeep Neema, Douglas C Schmidt, Aniruddha Gokhale, and Balachandran Natarajan. 2003. An approach for supporting aspect-oriented domain

modeling. In *International Conference on Generative Programming and Component Engineering*. Springer, 151–168.

[14] Giancarlo Guizzardi, Luis Ferreira Pires, and Marten J Van Sinderen. 2002. On the role of domain ontologies in the design of domain-specific visual modeling languages. In *Proc. 2nd Workshop on Domain-Specific Visual Languages*.

[15] Paul Hudak. 1997. Domain-specific languages. *Handbook of programming languages* 3, 39-60 (1997), 21.

[16] Gabor Karsai, Holger Krahn, Claas Pinkernell, Bernhard Rumpe, Martin Schindler, and Steven Völkel. 2014. Design guidelines for domain specific languages. *arXiv preprint arXiv:1409.2378* (2014).

[17] Dimitrios S Kolovos, Richard F Paige, Tim Kelly, and Fiona AC Polack. 2006. Requirements for domain-specific languages. In *Proc. of ECOOP Workshop on Domain-Specific Program Development (DSPD)*, Vol. 2006.

[18] Ákos Lédeczi, Arpad Bakay, Miklos Maroti, Peter Volgyesi, Greg Nordstrom, Jonathan Sprinkle, and Gábor Karsai. 2001. Composing domain-specific design environments. *Computer* 34, 11 (2001), 44–51.

[19] Jesús J López-Fernández, Jesús Sánchez Cuadrado, Esther Guerra, and Juan De Lara. 2015. Example-driven meta-model development. *Software & Systems Modeling* 14, 4 (2015), 1323–1347.

[20] Marjan Mernik, Jan Heering, and Anthony M Sloane. 2005. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)* 37, 4 (2005), 316–344.

[21] Bertrand Meyer. 2009. *Touch of Class: learning to program well with objects and contracts*. Springer Science & Business Media.

[22] Greg Nordstrom, Janos Sztipanovits, Gabor Karsai, and Akos Ledeczi. 1999. Metamodeling-rapid design and evolution of domain-specific modeling environments. In *Proceedings ECBS'99. IEEE Conference and Workshop on Engineering of Computer-Based Systems*. IEEE, 68–74.

[23] Puneet Patwari, Amar Banerjee, Subhrojyoti Roy Chaudhuri, and Swaminathan Natarajan. 2016. Learning's from Developing a Domain Specific Engineering Environment for Control Systems. In *Proceedings of the 9th India Software Engineering Conference*. ACM, 177–183.

[24] Ana Pescador and Juan de Lara. 2016. Dsl-maps: from requirements to design of domain-specific languages. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 438–443.

[25] Ana Pescador, Antonio Garmendia, Esther Guerra, Jesús Sánchez Cuadrado, and Juan de Lara. 2015. Pattern based development of domain-specific modelling languages. In *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 166–175.

[26] S Roy Chaudhuri, AL Ahuja, S Natarajan, and H Vin. 2009. Model-Driven Development of Control System Software. In *The Low-Frequency Radio Universe*, Vol. 407. 402.

[27] James Rumbaugh, Ivar Jacobson, and Grady Booch. 2004. *Unified modeling language reference manual, the*. Pearson Higher Education.

[28] Bran Selic. 2007. A systematic approach to domain-specific language design using UML. In *10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)*. IEEE, 2–9.

[29] Jonathan Sprinkle, Marjan Mernik, Juha-Pekka Tolvanen, and Diomidis Spinellis. 2009. Guest editors' introduction: What kinds of nails need a domain-specific hammer? *IEEE software* 26, 4 (2009), 15–18.

[30] Arie Van Deursen and Paul Klint. 2002. Domain-specific language design requires feature descriptions. *Journal of Computing and Information Technology* 10, 1 (2002), 1–17.